

Lecture 14: March 12

*Lecturer: Kathy Yelick**Scribe: Erika Chin*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

In these notes, we discuss Titanium, a partition-based global address space (GAS) language for parallel programming. The Titanium project was driven by the need for a way to adaptively parallelize increasingly complex models like tree-based data structures of hierarchical meshes and structured grids. In the last class, we saw that UPC is a GAS language in C. Titanium started as a similar project for C++, but it quickly evolved into a Java project due to the comparative ease of writing compilers for Java and its automatic memory management. Its features include multidimensional arrays, immutable classes, templates, operator overloads, SPMD parallelism, local and global references, global synchronization, and zone-based memory management. The code is compiled to C and then straight to machine code.

14.1 Titanium Execution Model

Titanium has an SPMD execution model (like UPC and CAF). In particular, Titanium uses:

- “Ti” to preface Titanium calls
- `Ti.thisProc()` to give the process ID
- `Ti.numProcs()` to give the number of total processes
- `Ti.barrier()` for global synchronization
- and other calls

There are two types of functions:

- Local: functions that can be independently called by any process
- Global: functions where all processes are called together

Other features:

- Barriers and single. When using barriers inside a method (to synchronize all processes), the keyword “single” is an optional annotation (for readability) that lets all processes call the method. If a barrier is used in a loop or branch, then single variables must also be used. Note: Every thread gets a copy of the single variable. At any time step, the variable will have the same value, but it is not the same variable.
- A broadcast will send a single variable to all other processes.
- Any possible instance of deadlock will be found at compile time.

14.2 Titanium Memory Model

Titanium does not have distributed arrays like UPC. See Slide 12 for a picture of the globally shared address space.

Model:

- References are either local (stored on that process) or global (stored on a remote process)
- Object heaps (references) are shared by default
- Program stacks are private
- Global references are more expensive than local references
- The compiler will try to infer local references when possible. Alternatively, the programmer can explicitly declare local references.
- Broadcasts will send references, not values. Be aware of possible data races.

Exchanges are all-to-all broadcasts. It takes the pointers from every process and broadcasts it to every other process. (This is similar to UPC's "directory" idea.) See Example on Slide 17. Note: It helps to read array declarations from right to left.

Typical garbage collection (GC) does not scale well in parallel cases. GC would have to be done across all processes, and it would have to check to see if an object has any global pointers pointing to it. Titanium implements region-based memory management which can achieve higher performance than typical garbage collection.

14.3 Support for Serial Programming

Features added to Titanium to support serial programming:

- Immutable classes - These are good when objects are small and will not be changed. Immutable classes prevent anyone from making assignments to fields outside of the constructors. They cannot inherit or be inherited. Semantically, it is a lot cleaner, and its performance is similar to the C struct performance.
- Operator overloading (like in C++)
- Multiple dimension arrays - These arrays allow the programmer to specify dimensions and index range of an array. It does indexing by Points, which is a tuple of ints. Subarrays of these arrays can be created without making additional copies. One can create different domains using union, difference, and intersection operators. One can refer to rows, columns, slabs, interiors, etc. Note: Domains give a shape (upper and lower bounds), not a space. Declare an array to create a space.
- Unordered iteration - This is useful for memory hierarchy optimizations and it allows the programmer avoid indexing details.
- Templates - Titanium provides template capabilities similar to C++ style templates.
- Cross-language calls - Titanium allows for calls to kernels and libraries in other languages. This enables programmers to write short, clean, modular code.

14.4 Performance and Applications

See graphs on Slides 42 and 43 for performance data. (The C code is the fastest. The yellow and green bars are the Titanium code.) There are many applications of Titanium: Poisson solvers for infinite domains, gas dynamics with AMR (adaptive mesh refinement), etc. Overall, Titanium allows for fewer lines of code with a comparable performance to C++.

14.5 Compiler and Language Status

Current status of Titanium:

- Titanium can run on almost any machine. It just needs a C compiler, C++ for the translator, pthreads for shared memory, and GASNet for distributed memory.
- It can run on a cluster.
- It can run on MPI, but it does not have good performance. It is just for portability.
- Work-in-progress: indexed array copy, non-blocking array copy, inspector/executor

Look on Slide 55 for possible CS267 project ideas and future work.